MAJOR TECHNICAL PROJECT

MAJOR TECHNICAL PROJECT

on

FLUID FLOW MODELING USING DEEP LEARNING

submitted by

PIYUSH BAFNA AND SHASHI KUMAR FAGNA

SCHOOL OF COMPUTING AND ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY. MANDI

ROLL NUMBER
B17020 - Piyush
B17141 - Shashi

YEAR OF SUBMISSION

2021

Certificate

It is certified that the work contained in the thesis titled **FLUID FLOW MODELING USING DEEP LEARNING** by **Piyush Bafna and Shashi Kumar Fagna** has been carried out under my/our supervision. .

Supervisor
Dr. Gaurav Bhutani
School of Engineering
Indian Institute of Technology. Mandi

DECLARATION BY THE CANDIDATE

We, **Piyush Bafna and Shashi Kumar Fagna**, certify that the work embodied in this report is not plagarised and carried out by me under the supervision of **Dr. Gaurav Bhutani** from **January 2020** to **May 2021**, at the School of Engineering, Indian Institute of Technology, Mandi. We declare that We have faithfully acknowledged and given credits to the research workers wherever their works have been cited in the work in this report. We further declare that we have not willfully copied any other's work, paragraphs, text, data, results, *etc.*, reported in journals, books, magazines, reports dissertations, theses, *etc.*, or available at websites and have not included them in this report and have not cited as my own work.

Date: June 14, 2021
Place: Mandi, India

CERTIFICATE BY THE SUPERVISOR

It is certified that the above statement made by the student is correct to the best of my/our knowledge.

Supervisor
Dr. Gaurav Bhutani
School of Engineering
Indian Institute of Technology. Mandi

Signature of Head of Department

Acknowlegement

We would like to sincerely thank Dr. Gaurav Bhutani for his guidance during the entire duration to make this project a great success. His inputs in this project have played major role in obtaining noval results.

We would also like to thank Dr. Kunal Ghosh and Dr.Aditya Nigam for carrying out the Major Technical Project processes in a very convenient manner and giving us their valuable feedback on work to help us improve.

We would also like to thank Mr. Hrishikesh Sagar for providing us with his previous work material and a helping hand in this entire project.

Contents

List of Figures

List of Tables

Abstract

Fluid flow simulation using traditional solvers based on Newtonian mechanics are computationally expensive because these require doing sequential calculations which are done using CPU. Our work aimed to decrease the computation time of these simulations using neural networks in a computationally expensive process of simulations. Our work was to implement the fluid simulation using deep learning methods. So, we did literature research and found a research paper based on the most recent graph-based deep learning approach. We replicated the graph model to validate the results provided in the paper and later on, fine-tuned it to make it more generalized for a variety of fluids.

Introduction

2.1 Fluid Flow Simulation

Fluid flow simulation is based on numerical analysis and mathematical modelling to solve and simulate the fluid flow in certain environments with constraints. Fluid Simulation plays an important role in modelling river flow, used in gaming for realistic effects, modelling chemical tanks that are in motion, etc. These simulations help to understand the fluid flow in certain types of products as well as help in predicting the effect of flow in certain environments. Hence, this works great for experiments that require a lot of resources and time when done practically.[3]

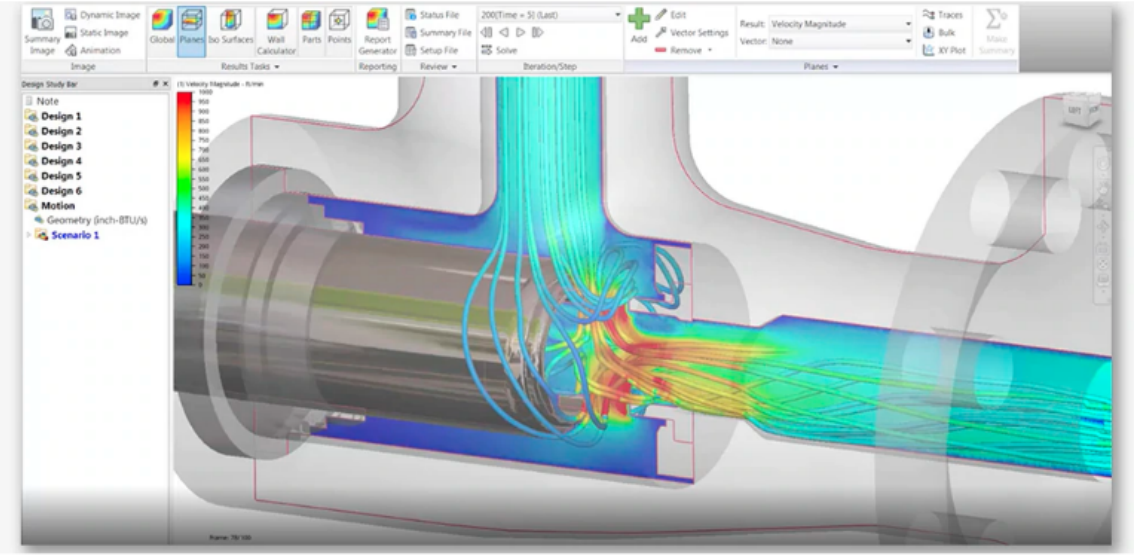

Figure 2.1: Fluid Simulation in Water Pump [3]

2.2 Material-Point-Method Technique

Material-Point-Method (MPM) is a numerical method technique used to simulate the motion of solids, liquids, and gases. In this method, the continuum body is considered as a collection of material points and each material point is surrounded by a background mesh that is used to calculate the gradient terms.

2.3 Importance of Deep Learning

Fluid flow Simulation is useful for modelling various situations. But the simulation uses numerical methods to calculate gradient terms such as deformation gradient etc, which requires a lot of computation power and time. Repetitive calculation results in increased computation which leads to a huge amount of time required for such simulation. Deep learning models can help perform these simulations in less time and computation. While training over huge amounts of data, deep learning models learn and form simpler mathematical formulas which can provide good quality of the simulation in a shorter amount of time. The efficiency of such a deep learning model can help to create high-quality simulation on portable, less powerful PCs as compared to High-Performance Cluster required for current techniques.

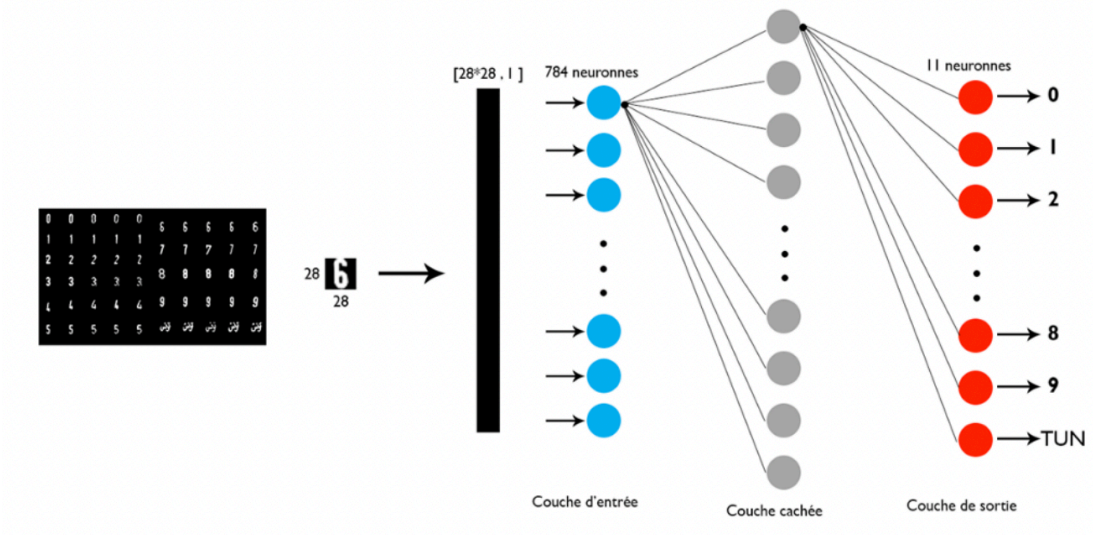

Figure 2.2: Example of Deep Learning Model [6]

Literature Review

Liquid splash modeling with neural networks results demonstrates the implementation of a deep learning model to improve visual fidelity with a large amount of realistic droplet formation and yields splash detail much more efficiently than finer discretizations. It is based on a data-driven approach to predict the formation of splashes and velocity changes. It consists of 2 Neural Network which indicator predict if particle form splashes (disconnected water region) in the next t. If so, velocity modification Δv will predict the change in velocity for a particular waterbody splash.[7] We use neural networks to model the regression of splash formation using a classifier together with a velocity modifier. Present model used Fluid Simulation Using Implicit Particles simulation for training, dataset and testing. The use of a simple Neural Network in this paper could not handle the interaction between particles and hence was primitive.[1]

GNS Framework

4.1 General Learnable Simulator

Let Xt be the state of the fluid at time t. The trajectory of states over k timesteps,

$$X^{t_{0:K}} = \left(X^{t_0}, \ldots, X^{t_K}\right) \tag{4.1}$$

A simulator, s:, models the dynamics by mapping the previous state to the current state. Simulated "rollout" trajectory is denoted as

$$\bar{X}^{t_{0:K}} = \left(X^{t_0}, \overline{X^{t_1}}, \ldots, \overline{X^{t_K}}\right) \tag{4.2}$$

which is computed iteratively by simulator s for each timestep. The current state is used by the simulator to compute the dynamics information for updating the current state to a predicted future state.

A learnable simulator, $\boldsymbol{s}_\theta$, computes the dynamics information with a parameterized function approximator, $d_\theta : \chi \rightarrow$ y, where parameter, θ, is learned. $Y \in$ y represents the dynamics information and the update mechanism determines its semantics. Here update function,

$$\overline{X^{t_{K+1}}} = Update\left(\overline{X^{t_K}}, d_\theta\right) \tag{4.3}$$

is an Euler integrator that takes the $\boldsymbol{X}^{t_k}$ and uses $\boldsymbol{d}_\theta$ to predict the next state.

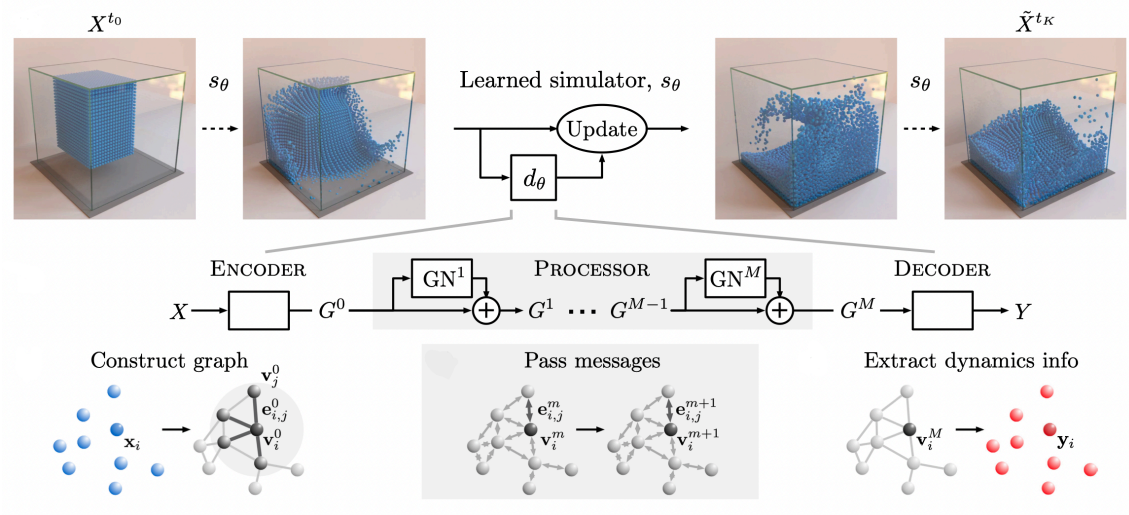

Figure 4.1: GNS Model Overview [2]

4.2 Simulation as a message-passing on a graph

This model adopts a particle-based representation of the physical system.

$$X = (x_0, \ldots, , x_N) \tag{4.4}$$

where each of the N particles' $\boldsymbol{x}_i$ represents its state. Physical dynamics of particles is approximated by an interaction like exchange of momentum and energy between two particles etc. The number of particle interactions decides the quality of the model.

We can define each particle as a node and pairwise interaction between them as edges of the graph. These interactions can be accounted using Encoder, Processor, and Decoder which together forms $\boldsymbol{d}$, parameterized function approximator.

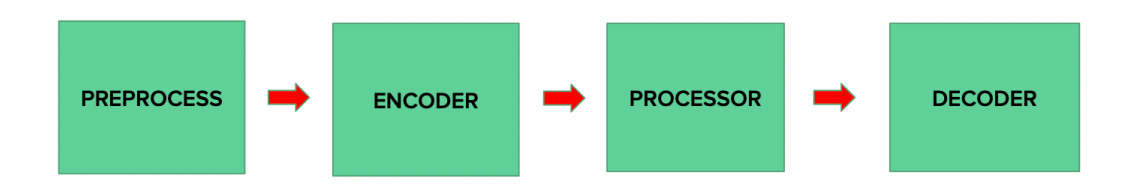

Figure 4.2: Model Architecture shown in form of blocks

4.2.1 Encoder Definition

Encoder $\chi \rightarrow \mathcal{G}$ embeds the particle's state and interactions into a latent graph

$$G^0 = \text{Encoder}(X) \tag{4.5}$$

where $G = (V, E, u), v_i \in V, e_{i,j} \in E$. The node embeddings, $v_i = \varepsilon^e\left(x_i\right)$, are learned functions of the particles' states. The edge embeddings, $e_{i,j} = \varepsilon^e\left(r_{i,j}\right)$, are learned functions of the pairwise properties of the corresponding particles $r_{i,j}$.

For example - spring constant, the displacement between particles, etc. The graph-level embedding, $\boldsymbol{u}$, can represent global properties such as gravity and magnetic fields.

4.2.2 Processor Definition

Processor $\mathcal{G} \rightarrow \mathcal{G}$ processes over latent graph's nodes and edges to predict future interaction. This processing over $\boldsymbol{m}$ times is called message-passing, to generate a sequence of the updated latent graph,

$$G = \left(G^1, \ldots, G^M\right) \tag{4.6}$$

where $G^{m+1} = GN^{m+1}\left(G^m\right)$

This message-passing updates the latent graph keeping all the constraints in context.

4.2.3 Decoder Definition

From the node of the final latent layer, DECODER : $\mathcal{G} \rightarrow \mathcal{Y}$, dynamics information is extracted by

$$\mathbf{y}_i = \delta^v\left(\mathbf{v}_i^M\right) \tag{4.7}$$

Learning δ^v should cause the $\boldsymbol{y}$ representations to reflect relevant dynamics information, such as acceleration, in order to be semantically meaningful to the update procedure.

Training Data

5.1 Data Generation

We used manta and ligghts for generating the dataset. Respective script was developed for both manta and ligghts to generate the water dataset for accounting about - of of particle in a simulation at maximum. Timestep for the same was set as -ms. And - number of flames were generated in sample data for training data purpose.

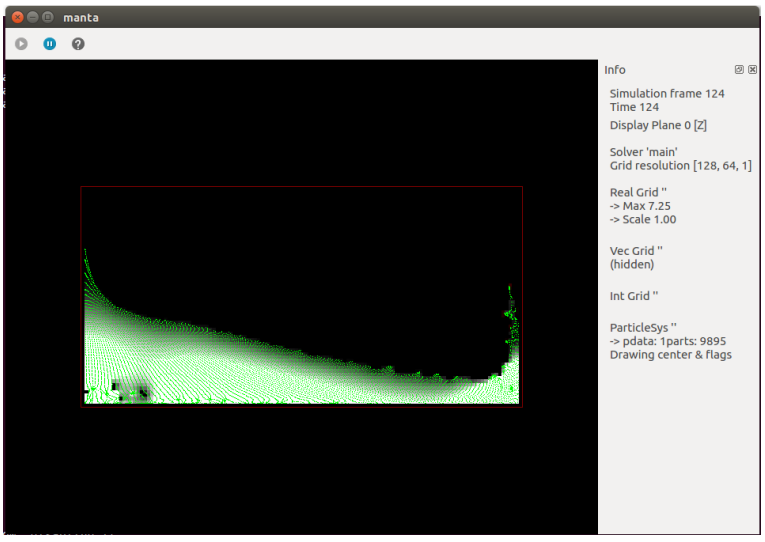

Figure 5.1: Data genration using Mantaflow

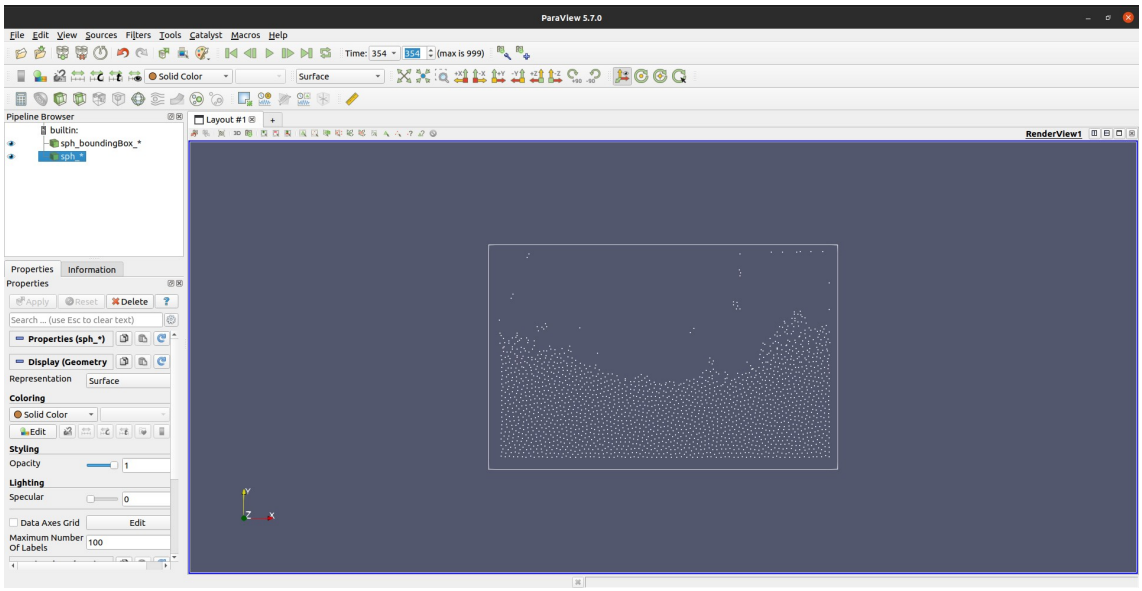

Figure 5.2: Data generation using LIGGGHTS

5.2 Data Available

We have also used the generated datasets provided by the Deep Mind Research Organisation. For our training purposes, we used Goop, Sand and WaterRamps Datasets. Each Dataset consists of 4 files, Training set, Validation set, Test set and metadata.json file. The metadata.json file contains the essential information about particle type, number of timesteps, boundary box etc. The data files are provided in tf.records format.

Domain	Max Particles	Trajectory Length	Δt
SAND	2k	320	2.5 ms
GOOP	500	600	2.5 ms
WATER	5.8k	800	2.5 ms
WATER-3D	13k	800	5 ms
SAND-3D	20k	350	5 ms
GOOP-3D	14k	300	5 ms
WATER RAMP	2.3k	600	5 ms
FLUIDSHAKE-BOX	1.4k	1500	2.5 ms
WATERDROP	0.6k	1k	5 ms
FLUIDSHAKE	1.3k	2000	2.5 ms
MULTIMATERIAL	1.6k	1000	2.5 ms

Table 5.1: Information about the datasets

Generating and Preprocessing Dataset

6.1 Generating Dataset

To move towards training our model on real world datasets, our next step was to create our own dataset. Softwares like like Ligggths and Mantaflow were used to create data for our experiments.

6.1.1 Generating Data using Mantaflow

Mantaflow is an open-source framework used for fluid simulation research in Computer Graphics and Machine Learning. It has a parallelized C++ solver core, python scene definition interface and plugin system that allows quick prototyping and testing of new algorithms. A wide range of Navier-Stokes solver variants are included to simulate the flow. Our simulations are generated using the Mantaflow framework. Some features of the mantaflow are as follows-

- Eulerian simulation using MAC Grids, PCG pressure solver and MacCormack advection.
- Flexible particle systems
- FLIP simulations for liquids
- Surface mesh tracking
- Free surface simulations with levelsets, fast marching.

To generate data using Mantaflow following steps were followed:

1. Build Manta on your systems following the documentation available on the official site of Mantaflow.
2. Run the script *mtp_data_gen.py* from the manta directory using the command *manta mtp_data_gen.py*

3. A new folder under name ”Data” is created. This folder contains *x_x_x.txt* files of each frame of all the samples. The *x_x_x.txt* files contains numpy arrays.

4. Once the data is generate, now use the script *raw.py* to sort the files in their respective samples.

5. Next, run the script *convert_to_tfrecord.txt* to convert the data into tfrecord format.

6. A new file is created under the name *test.tfrecord*. This file contains all the data and can be directly used to train or test the model.

6.1.2 Generating Data using LIGGGHTS

LIGGGHTS is an Open Source Discrete Element Method Particle Simulation Software. LIGGGHTS stands for LAMMPS improved for general granular and granular heat transfer simulations. LAMMPS is a classical molecular dynamics simulator. It is widely used in the field of Molecular Dynamics.[5] Our simulations are generated using the LIGGGHTS framework. Some features of the LIGGHTS are as follows:-

- An easy-to-understand structure of the code.
- A flexible scripting language allowing efficient usage.
- Efficient parallelization.
- The basics of the GRANULAR package, like particle insertion, shear history implementation.
- Post-processing.

To generate data using LIGGGHTS following steps were followed:

1. Build LIGGGHTS on your systems following the tutorial available on the EngineerDo website [4].

2. Run the created script *in.mtp_data_gen* from the LIGGGHTS directory using the command *mpirun -np 8 lmp380 -in in.mtp_data_gen*

3. A new folder under name ”post” is created. This folder contains *x_x_x.vtk* files of each frame of all the samples. The *x_x_x.vtk* files contains numpy arrays.

4. Once the data is generate, now use the script *raw.py* to sort the files in their respective samples.

5. Next, run the script *convert_to_tfrecord.txt* to convert the data into tfrecord format.

6. A new file is created under the name *test.tfrecord*. This file contains all the data and can be directly used to train or test the model.

6.2 Preprocessing

Understanding of structure would help us in using the same model for real world data simulation as well as modifying the given dataset. So far the data available to us was in tfrecords format. Our task was to understand how the data was encoded into tfrecords. The paper doesn't mention anything about data generation. So we had to do reverse engineering to figure how the data was pre-processed before converting into tfrecords

The decoding could not be done until the Feature Description of the dataset was identified which we were able to identify the feature description from the rollout and training script.

The data was encoded in two steps that is Feature Description and Context Features.

Feature Description contains the positions of particles of each frame appended one after the other.

Context features contains two placeholders: Key and ParticleType. Key denote the sample number and ParticleType denotes the types of particles present in the sample.

Experiments

7.1 Experiment 1: Creating Specific Models for Sand, Goop and Water Ramps.

In this experiment, we trained separate models each for Sand, Goop and Water Ramps. First, we trained a model using Water-Ramps dataset and used the weight of this model to train other models for sand and Goop using the same training parameters.

One-step Loss : Loss of one iteration of a training sample is called one-step loss.

Rollout Loss : Loss of all the frames of a training sample is termed as rollout loss.

7.1.1 Water-Ramps

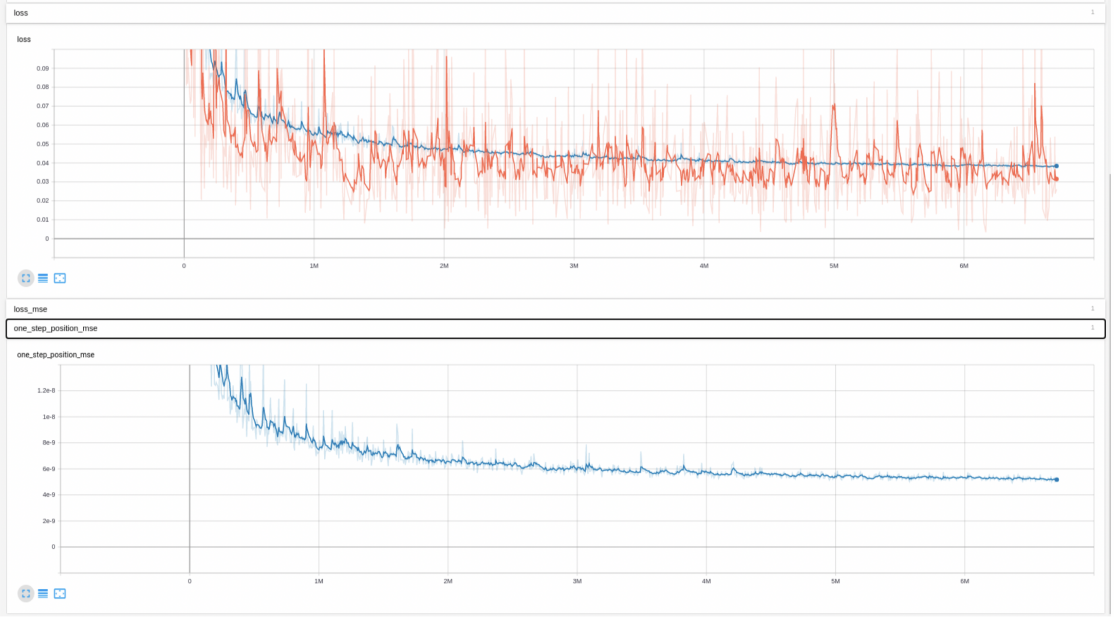

Figure 7.1: WaterRamps : (a) Training Loss vs training Steps. (b) Validation Loss

Domain	Our One-Step Test loss per particle(10^{-9})
WATER-RAMPS	6.00
SAND	9.10
GOOP	5.00

Table 7.1: One step Test loss obtained for WaterRamps, Sand and Goop

Domain	Nodes(N)	Edges(K)	Training Steps
WaterRamps (MPM (2D))	2.3k	600	8M
SAND	2k	320	3M
GOOP	1.90	400	3M

Table 7.2: Material type, Nodes(N), Edges(K), Training steps

Rollout Snapshots from Water-Ramps

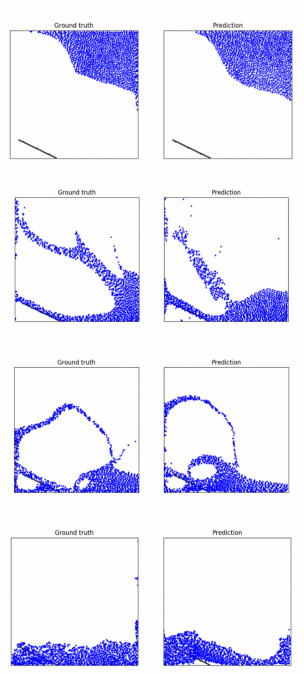

Figure 7.2: Rollout from Trained Water-Ramps Model

Observations

We observe a gradual decrease in validation loss over the training process and the best validation loss is 5×10^{-9} (one step MSE loss).

7.1.2 Goop

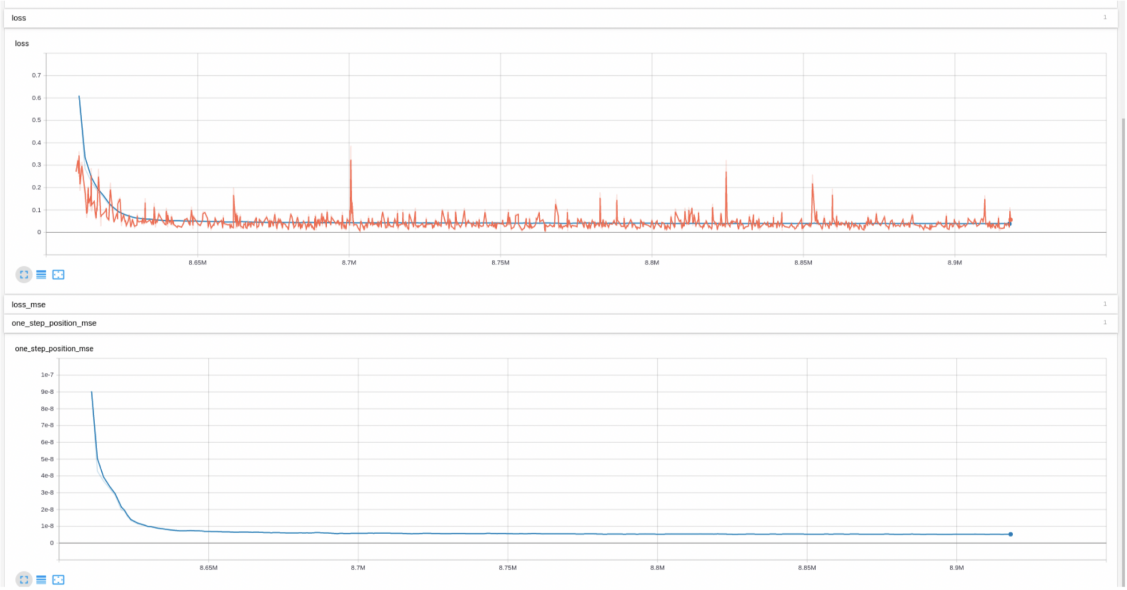

Figure 7.3: Goop: (a) Training Loss vs training Steps. (b) Validation Loss. Plot in Orange : Training Graph, Plot in Blue : Validation Graph

Rollout Snapshots from Goop

Figure 7.4: Rollout from Trained Goop Model

Observations

We observe a gradual decrease in validation loss over the training process and the best validation loss is 3×10^{-9} (one step MSE loss).

7.1.3 Sand

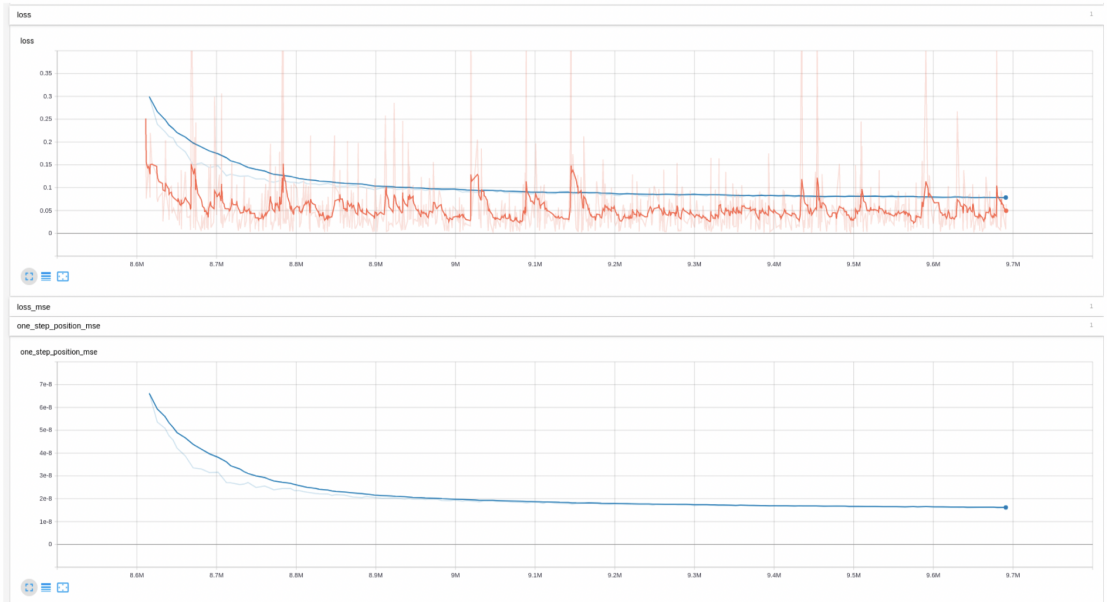

Figure 7.5: Sand : (a) Loss vs training Steps. (b) Loss vs Validation Steps.Plot in Orange : Training Graph, Plot in Blue : Validation Graph

Rollout Snapshots from Sand

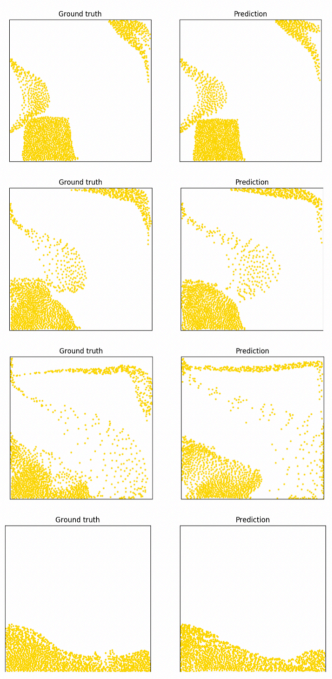

Figure 7.6: Rollout from Trained Sand Model

Observations

We observe a gradual decrease in validation loss over the training process and the best validation loss is 7.09×10^{-9} (one step MSE loss per particle).

7.1.4 Comparison of Our Model with the Given Model

Material	Given One-Step Test Loss(10^{-9})	Our One-Step Test Loss(10^{-9})	Given Rollout Test Loss(10^{-3})	Our Rollout Test Loss(10^{-3})
WaterRamps (MPM (2D))	4.91	6.00	3.34	5.64
SAND	6.23	9.10	2.37	4.6
GOOP	2.91	5	1.89	2.7

Table 7.3: Comparison of our trained model to the given model

7.1.5 Conclusions

- In WaterRamp, we trained our model over ramps of different length and slopes. Each training data in the dataset contains different kinds of ramps.
- If we train a specific model over the weights of a pretrained model over another dataset, we observe that the convergence occurred at very less train steps as compared to training it from scratch.
- When we train models over 200k training steps, prediction results work but are not accurate. The accuracy can be increased by increasing the number of training steps upto 3 million steps.
- We understood that the training time of the model depends on the number of Nodes and Edges in the graph model of that particular dataset.
- We were able to achieve values close to the values given in the research paper. The 10 avg difference between our loss and given loss is 1.78×10^{-3} in case of rollouts and 2.01×10^{-9}.

7.2 Experiment 2: Creating a Generic Model for materials that have similar properties.

In this experiment, our objective was to obtain a generic model for the particles that have similar properties. From Table 6.4, we observed that sand and goop have similar properties in terms of

number of nodes and Edges, making graph structure almost similar . Hence it is possible to obtain a generic model that predicts the behavior of both materials.

Domain	Nodes(N)	Edges(K)
Sand (MPM (2D))	2k	320
Goop (MPM (2D))	1.9k	400

Table 7.4: Material type, Nodes(N), Edges(K)

In order to obtain a generic model, we fine-tuned a model trained on a similar task. In this case, we first trained a model on the sand and then fine-tuned it on goop. For fine-tuning, we used the weights of the trained model and freeze the Encoder layers and GNS framework, and unfreeze the decoder part. Therefore, only the unfreezed layers are trained. The learning rate is also lowered in this case.

The model is trained for the first 6M steps on the sand and then fine-tuned on Goop Dataset for 2M steps.

7.2.1 Analysis of Plots Obtained for Generic Model

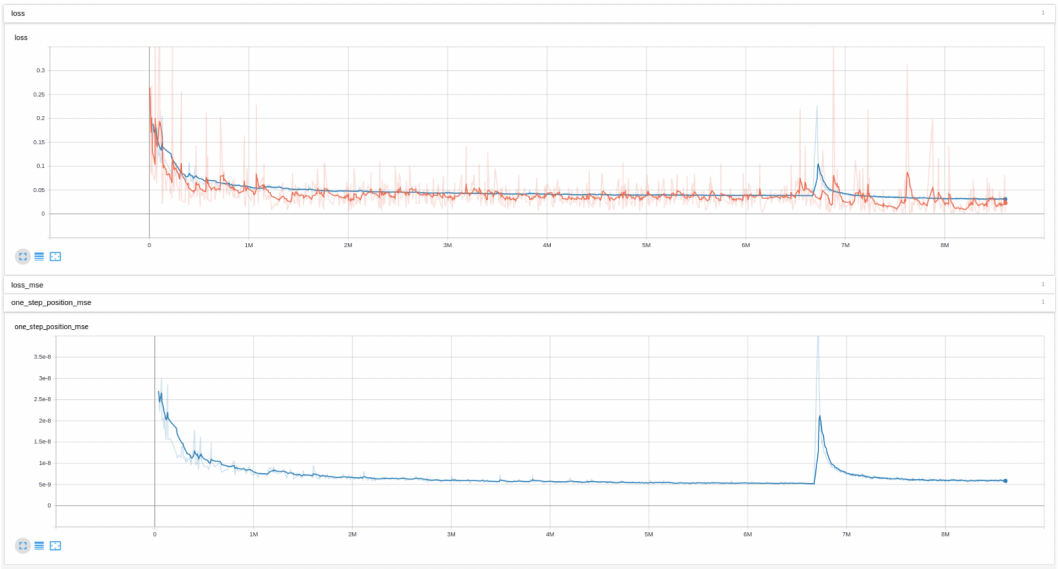

Figure 7.7: Generric Model: (a) Loss vs training Steps. (b) Loss vs Validation Steps.

7.2.2 Observations

High loss is observed at the start of the training process. The loss gradually decreases till the 6M training step. A second peak is observed after 6M, that is at the start of fine-tuning on Sand Dataset. Hence, we observed that model when trained for two dataset forgets the initial dataset learnings. The model behaves randomly on the initially trained dataset.

Performance of model

Material Type	Our one-step loss $\left(10^{-9}\right)$
Goop	7.23

Table 7.5: Performance when model was trained only on Goop

Material Type	Our one-step loss $\left(10^{-9}\right)$
Goop	∞ wrt to Sand
Sand	3.00

Table 7.6: Performance when model was trained on both Sand and Goop

7.2.3 Conclusion

To generalize, the model was trained over one type of dataset(X) for 80% of the training time and remaining time on another dataset(Y). This makes the model behave more like that it is trained on a dataset(Y) than on both X and Y. No improvements were found even after changing the learning rate, training steps, or training dataset.

7.3 Experiment 3: Creating a Generic Model for materials that have similar properties using other approach

In the previous experiment of generalization, we followed the method of sequential training. The results obtained were not desirable. We observed that the model when trained for two dataset forgets the initial dataset learnings. The model gives out completely random predictions for the initially trained dataset. The performance of model could not be improved even after varying the learning rate, increasing the number of training steps.

In order to achieve our objective of making the model generic, we then used the multimaterial dataset. This dataset consists of multimaterial samples as well as single material samples all put

together.

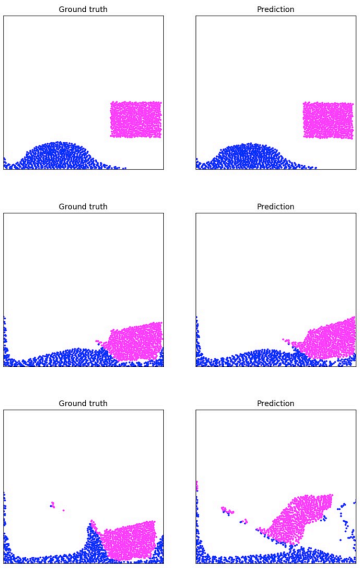

Figure 7.8: Rollout of Goop-Water combination

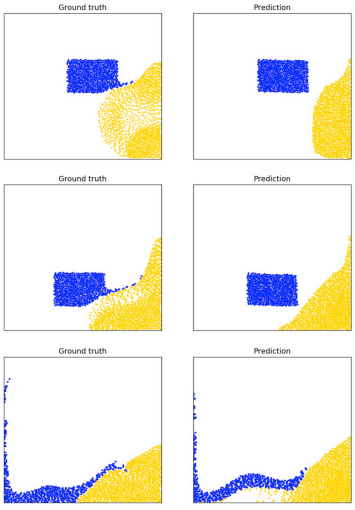

Figure 7.9: Rollout of Water-Sand combination

7.3.1 Observations

- The model was able to generalise. It was able to generate good results for different kinds of materials individually. The results are shown in Fig 6.9, 6.10.
- The model was also able to generate good results for multi-materials. It model was able to learn the interaction between the different types of particles. as shown in Fig 6.9, 6.10.

- The model was able to learn the properties of materials like density, viscosity etc.

7.4 Experiment 4: Scalabilty (Downscale)

The aim of this experiment was to analyse the performance of the model on lesser number particles which was trained on large number of particles. In this experiment, we trained a model using WaterDrop dataset which consists of around 13,000 particles per sample. This model was then tested on dataset consisting of around 1000-2000 particles per sample.

The dataset with reduced number of particles was generated by altering the available dataset using the generating and pre-processing techniques mentioned in Generation and Preprocessing section.

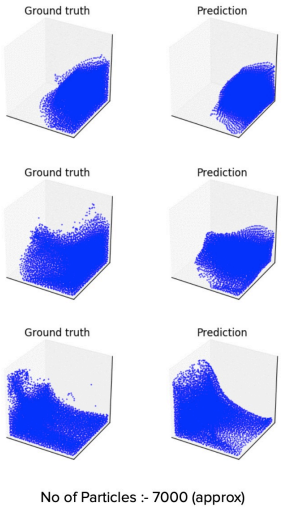

Figure 7.10: Rollout from model trained on WaterDrop Dataset with 13k particles

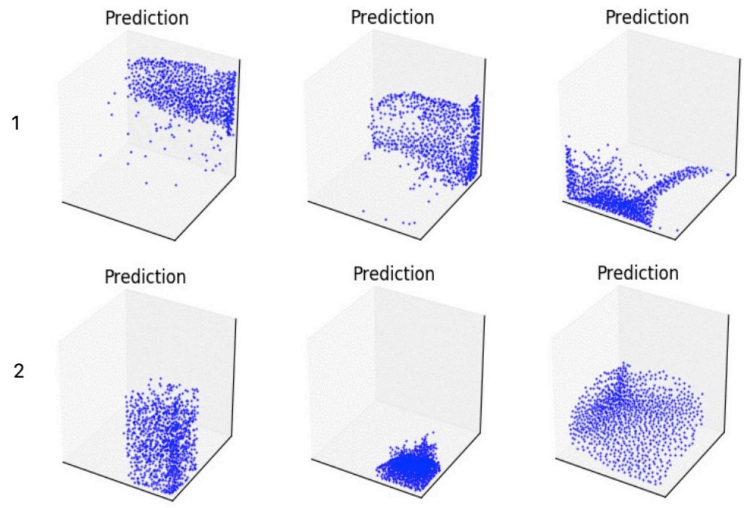

Figure 7.11: Rollouts from previously trained Model on down scaled test dataset with 2k particles

7.4.1 Observations

We observed that the model which was earlier trained on set containing large number of particles was able to generate good results even on lesser number particles.

7.5 Experiment 5: Scalability (Up scaling)

In previous experiment we analysed the performance of model on down scaled data. Our next step was to check its performance on the up scaled data. For this experiment, we generated our own dataset using the methods mentioned in the section of Generating and Preprocessing Datasets. The training dataset consists of the around 500-800 particles per sample. Each sample contains only 200 frames. For Testing the data, we generated multiple test datasets with different number of particles. In order to control the number of particles, we adjusted the radius as per our convenience. We used dataset with radius 0.16 for training and radius of 0.20mm, 0.30mm 0.50mm 0.70mm for testing.

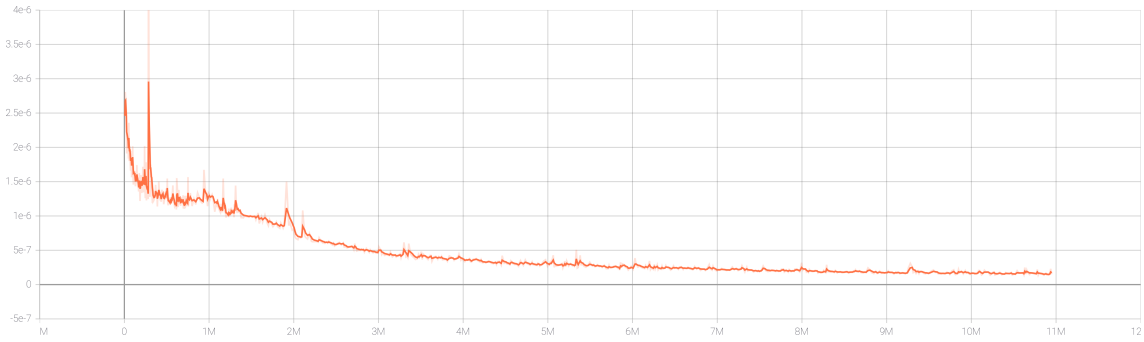

Figure 7.12: Validation Loss vs Training Steps

The best validation loss of 1.16 was obtained after 4M training steps.

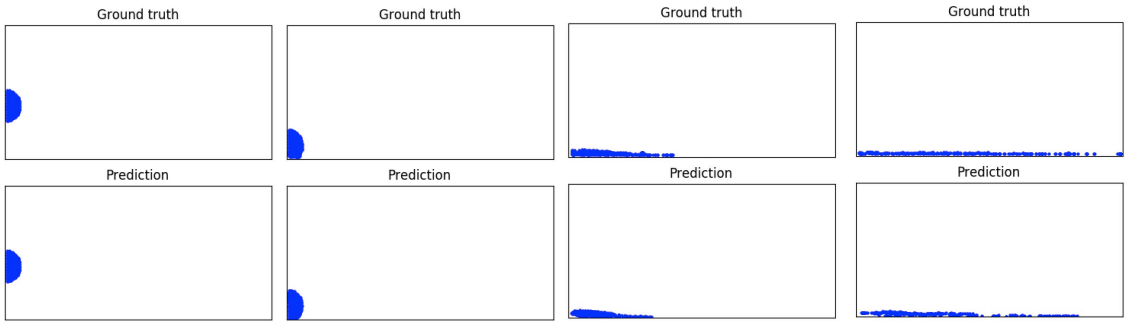

Figure 7.13: Rollout of model trained on 500-800 particles per frame.

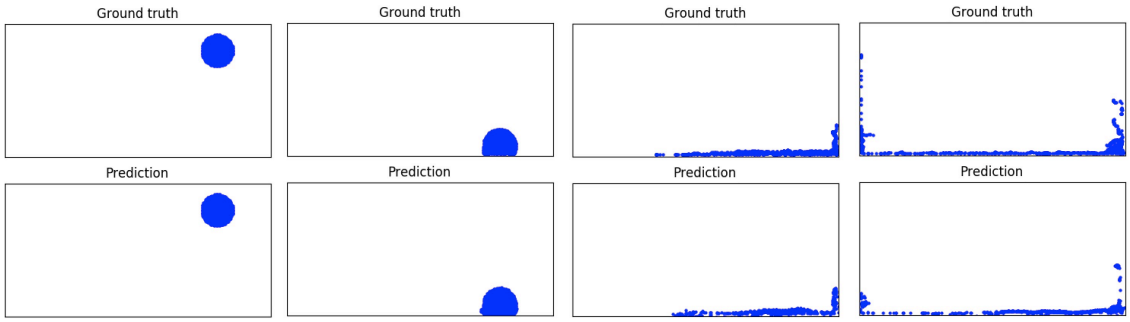

Figure 7.14: Rollout of previous trained model on test dataset with 1500 particles

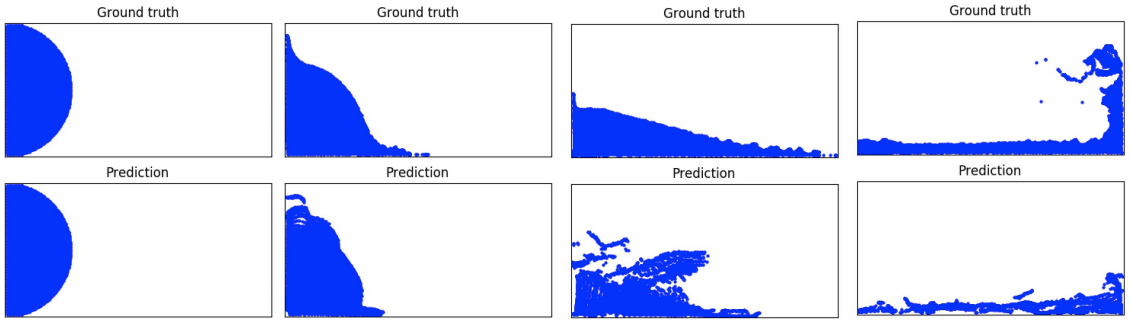

Figure 7.15: Rollout of previous trained model on test dataset with 17000 particles

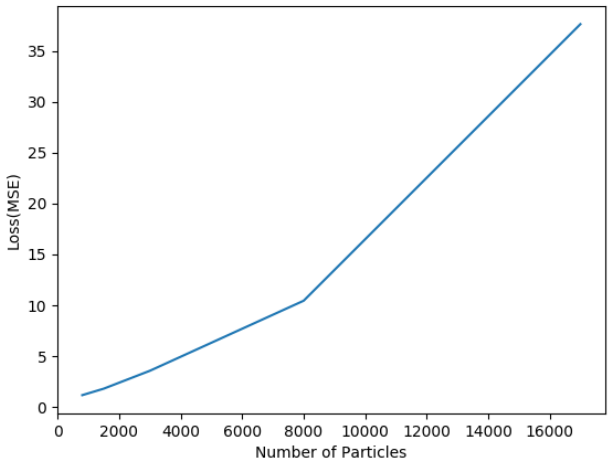

Figure 7.16: Loss Vs Number of Particles

7.5.1 Analysis

- We observed that the average prediction generation time is always less than the average ground truth generation. This means that the Deep Learning model is able to generate faster results as compared to simulating softwares.

Radius Size(mm)	Approx. Number of Particles	Loss MSE
0.16	500-800	1.16
0.2	1500	1.80
0.30	3000	3.56
0.50	8000	10.45
0.7	17000	37.66

Table 7.7: Radius size(R), Number of particles(N), Loss (MSE)

Approx. Number of Particles	Avg. Ground Truth Generation Time	Prediction Generation Time
500-800	10 secs	5.4 secs
1.5k	15 secs	7.1 secs
3000	22 secs	10 secs
8000	140 secs	45 secs
17000	310 secs	65 secs

Table 7.8: Radius size(R), Number of particles(N). Each sample contains 200 frames

- The loss increases in exponential manner as the the number of particles are increased. This means that the model makes too much of approximation as the number of particles is increased.
- Though the loss is higher in case of samples with large number of particles, the overall trajectory seems to be realistic.

7.6 Experiment 6: Free objects

In experiment 1, we used static objects i.e ramps. In this experiment, we used free objects like a solid box. The dataset contains samples where the container with water and the box was shaken side to side as shown in the figure 7.17

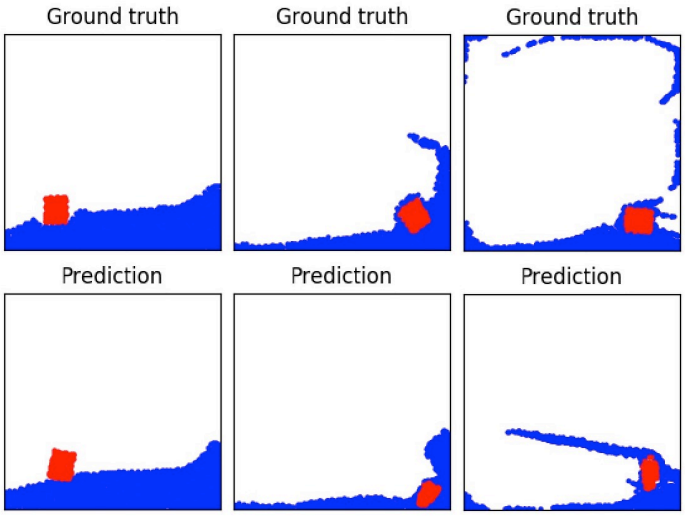

Figure 7.17: Rollout of trained model on FluidShake-Box dataset

7.6.1 Observations

- The model was able to predict the motion of the rigid block accurately.
- Though the particles of the rigid block remain intact with each other, the shape of the block is deformed.

.

Conclusions

1. **Takes Lesser time to generate simulations**

From the experiments, it is evident that the time taken to generate the predictions is very less compared to that of simulation softwares like Mantaflow and Ligggths. This result is of great importance as it reduces the time taken to generate the simulations. Hence, we can generate many simulations using the Deep learning method as compared to the simulation softwares in a given amount of time as shown in table 7.8

2. **Generic to Particle Type**

From experiment 4, we conclude that a model once trained on different types of particles, can very well predict the motion of particles. Hence, a single model can be used to generate simulations of different kind of materials.

3. **Robust to obstacles**

From experiments 1 and 6, we conclude that the model is robust to both static obstacles as well as free objects. In case of free objects, we observe that the shape of the rigid box is deformed. However, the particles remain intact therefore the motion of centre of mass is unaltered.

4. **Capable of learning Material Properties**

From experiment 3, we conclude that the model was able to learn the properties of materials like the sand is denser than the water, goop is sticky material. Hence, the model learns the properties of material on its own.

5. **Independent of the number of particles**

From experiment 6, we conclude that the model can very well predict in case of down scaled

data. In case of upscaled data we see even though the loss is high, the overall trajectory of the material is quite similar to ground truth. This is a very interesting result as it gives us freedom to run the model on any data irrespective of the number of particles.

Comparing the model with existing DL models

The GNS model out performs all the existing models used for simulating fluid flow. The model uses state-of-the-art technique of Graph Neural Networks.

We implemented the CConv model, loss and training procedure as described by Ummenhofer et al. (2020) [8]. CConv's performance is best when modelling water-like fluids, which is what it was used to simulate in the original work. However, it still did not beat our GNS model, and it was clearly not as powerful for other materials and interactions between other materials. This is not particularly surprising, given that our GNS is a more general model, and our neural network implementation has higher capacity on several axes, e.g., more message-passing steps, pairwise interaction functions, more flexible function approximators.

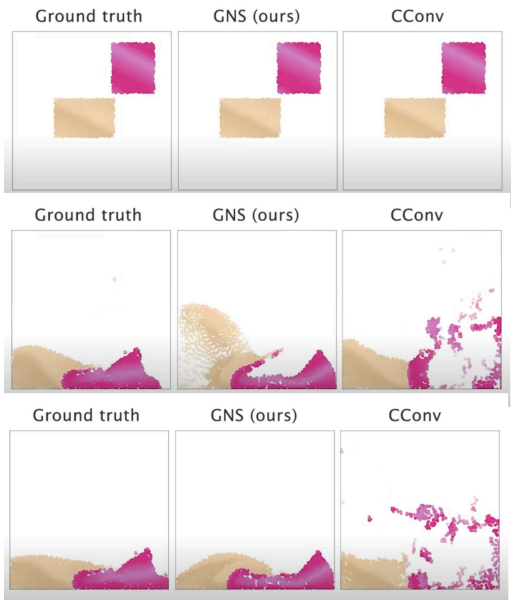

Figure 9.1: Actual vs GNS vs CConv

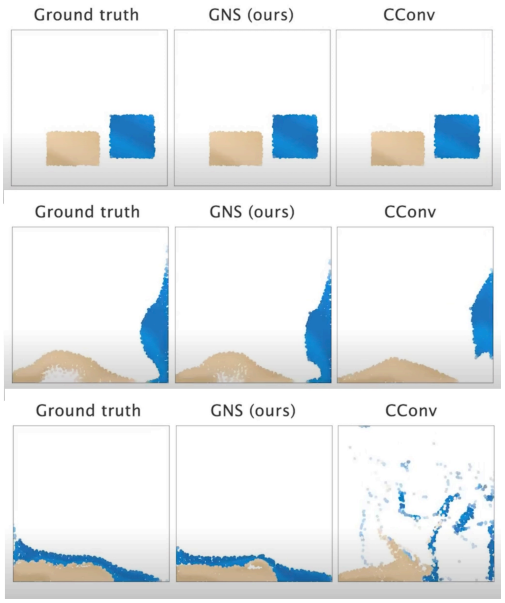

Figure 9.2: Actual vs GNS vs CConv

CConv can learn to replicate sand quite effectively, according to our qualitative results. However, it had trouble simulating materials with more complicated fine-grained dynamics. CConv mimicked the fluid well in the BOXBATH domain, but it failed to retain the box's shape. In the GOOP domain, CConv could not hold goop bits together and deal with the rest state, whereas MULTI-MATERIAL displayed local "explosions," where patches of particles abruptly burst outward.

Failure cases

In our series of experiments, we noticed two failure cases.

1. In case of FluidShake-Box Dataset, we observed that though the model was ableto predict the motion of the rigid box but the shape of the boxx was deformed towards the end of the trajectory. We speculate that the reason for this is that GNS has to keep track of the block's original shape, which can be difficult to achieve over long trajectories given an input of only 5 initial frames.

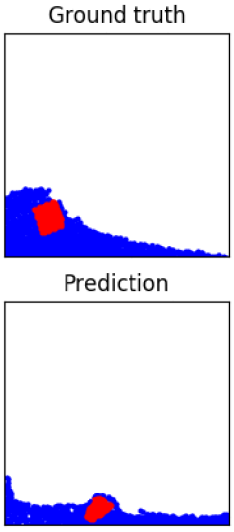

Figure 10.1: Failure in retaining box shape

2. The second failure cases was noticed in case of Goop Dataset. We observe that for some predictions, the blobs of goop material stickto the wall of the container. We speculate that the complexity of static friction and adhesion may be hard to learn. However, this can be overcome by exposing the model to more falling vs sticking phenomena.

References

[1] Wikipedia contributors. *Magnetohydrodynamic drive*. https://en.wikipedia.org/w/index.php?title=Magnetohydrodynamic_drive&oldid=993686041. [Online; accessed Feb. 14, 2021]. 2021.

[2] Miles Cranmer et al. "Discovering symbolic models from deep learning with inductive biases". In: *arXiv preprint arXiv:2006.11287* (2020).

[3] *Fluid Flow Flow Analysis Software*. https://www.autodesk.com/solutions/simulation/cfd-fluid-flow. [Online; accessed Feb. 14, 2021]. 2021.

[4] Sandia National Laboratories. *LIGGGTHS Simulator*. https://www.cfdem.com/. [Online; accessed 22-May-2021]. 2021.

[5] *LIGGGHTS OPEN SOURCE DEM PARTICLE SIMULATION CODE*. https://www.cfdem.com/liggghts-open-source-discrete-element-method-particle-simulation-code. [Online; accessed April. 18, 2021]. 2021.

[6] H. Sagar. ""Neural Network in snow avalanche dynamics"". In: Indian Institute of Technology Mandi. 2020.

[7] Kiwon Um, Xiangyu Hu, and Nils Thuerey. "Liquid splash modeling with neural networks". In: *Computer Graphics Forum*. Vol. 37. 8. Wiley Online Library. 2018, pp. 171–182.

[8] Benjamin Ummenhofer et al. "Lagrangian Fluid Simulation with Continuous Convolutions". In: *International Conference on Learning Representations*. 2020.

Appendix

A1 Application of fluid dynamics in Electrical Engineering

The **electronic–hydraulic analogy** is the most widely used analogy for "electron fluid" in a metal conductor. Since electric current is invisible and the processes in play in electronics are often difficult to demonstrate, the various electronic components are represented by hydraulic equivalents. [5]

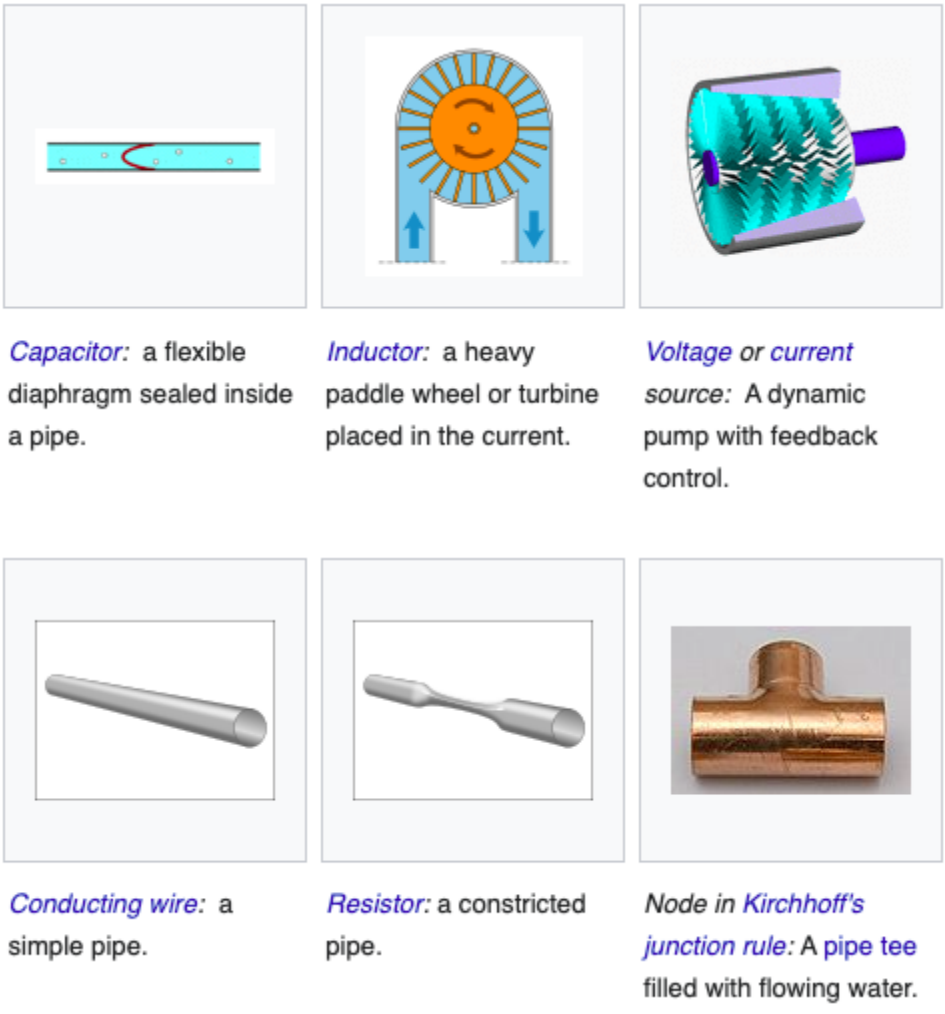

Fig [11] Examples of electronic components as Hydraulic equivalents

A **magnetohydrodynamic drive** or MHD accelerator is a method for propelling vehicles using only electric and magnetic fields with no moving parts, accelerating an electrically conductive

propellant (liquid or gas) with magnetohydrodynamics. The fluid is directed to the rear and as a reaction, the vehicle accelerates forward.

The working principle involves the acceleration of an electrically conductive fluid (which can be a liquid or an ionized gas called a plasma) by the Lorentz force, resulting from the cross product of an electric current (motion of charge carriers accelerated by an electric field applied between two electrodes) with a perpendicular magnetic field. The Lorentz force accelerates all charged particles (positive and negative species) in the same direction whatever their sign and the whole fluid is dragged through collisions. As a reaction, the vehicle is put in motion in the opposite direction. [6]

These two applications prove that fluid simulations are essential for understanding the electrically conductive fluid flow in Electrical Engineering.

A2 Setting up HPC for first time

1. To run models on HPC, login into your HPC account.
   ssh username@10.8.1.20
2. Once logged in, follow the below step to establish a python 3 environment on your HPC account. Open the .bash_profile file
   nano ~/.bash_profile
3. Add the following line in the file
   scl enable rh-python36 bash
4. Save the file
   source ~/.bashrc
5. Open the .bashrc file
   nano ~/.bashrc
6. Add the following line in the file
   export CUDAHOME=/opt/soft/share/cuda-10.0/
   export PATH=/opt/soft/share/cuda-10.0/bin ${PATH:+:${PATH} }
   export LD_LIBRARY_PATH=/opt/soft/share/cuda-10.0/lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH} }
   export CUDA_LIB_PATH=/opt/soft/share/cuda-10.0/lib64/stubs/
   export PATH=/opt/soft/share/cudann10/bin ${PATH:+:${PATH} }
   export LD_LIBRARY_PATH=/opt/soft/share/cudann10/lib64
   ${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH} }
   export PATH=. local /bin ${PATH:+:${PATH} }
7. Save the file
   source ~/.bashrc
8. Download latest get-pip.py from internet and transfer it into HPC file storage
   scp -r <location-on-pc> username@10.8.1.20:~/
9. Run the following command to install pip
   /opt/rh/rh-python36/root/usr/bin/python3 get-pip.py --user --proxy=http://10.8.0.1:8080
10. Install virtualenv
    pip3 install --user --upgrade --proxy=http://10.8.0.1:8080 virtualenv
11. Create a virtualenv
    virtualenv ~/wd/tf --python=/opt/rh/rh-python36/root/usr/bin/python3
12. Enter into virtualenv
    source ~/wd/tf/bin/activate
13. Install all the libraries in the virtualenv required for running model.
    pip install --proxy=http://10.8.0.1:8080 <package-name>

Source Credits : Dakash Thapar, PhD scholar IIT Mandi

A3 Setting up Anaconda on HPC if python doesn't work

In some nodes for example n72, n66 and n85 the python 3.6 package is not installed properly. This may hinder your training process.

The problem is sorted out by creating a virtual environment using Conda.

1. Download Conda, [7]
   apt-get install libgl1-mesa-glx libegl1-mesa libxrandr2 libxrandr2 libxss1 libxcursor1 libxcomposite1 libasound2 libxi6 libxtst6
2. Install Conda, [8]
   bash ~/Downloads/Anaconda3-2020.02-Linux-x86_64.sh
3. Create Virtual environment, [9]
   conda create -n yourenvname python=x.x anaconda
4. Activate virtual environment, [9]
   source activate yourenvname

A4 Submitting job on HPC for training models

1. Make the job file. Template is provided below.

```
#!/bin/bash
#PBS -q gpuq
#PBS -o out.o
#PBS -e out.e
#PBS -N myjob
#PBS -j oe
#PBS -l nodes=1:ppn=1
#PBS -V
cd ${PBS_O_WORKDIR}
echo "Running on: "
cat ${PBS_NODEFILE}
cat $PBS_NODEFILE > machines.list
echo "Program Output begins: "
source ~/wd/tf/bin/activate
python filename.py
```

2. You can change number of node needed for the job by changing
   #PBS -l nodes=1:ppn=1

3. Submit job
   qsub job.sh

A5 Useful command for using HPC

1. qstat -a
   returns a list of all queued jobs.
2. qdel <job-id>
   stop job working on HPC.
3. qstat <job-id>
   return stat of job.
4. cat <file>
   open file in terminal.
5. cd ..
   navigate back to the previous folder.
6. ls -a
   Shows all folders (hidden include).
7. scp -r <source-location> <destination-location>
   copy file from source to destination.
8. scp <source-location> <destination-location>
   copy folder from source to destination.

Source Credits : HPC website(Institute access only), IIT Mandi